# Proyecto 2: Mapa inteligente

Marcelo Vidal y Fabian Garcia.

Ingeniería Civil en Informática, Universidad Católica de Temuco.

INFO 1139, Programación de Robot.

Victor Valenzuela Diaz.

Martes 4 de Junio del 2024.

## Introducción.

Este proyecto tiene como objetivo principal la implementación de un sistema de mapa inteligente para ser recorrido por un robot y que este vaya descubriendo el recurso escondido en cada tile. Además la solución para mostrar un mensaje final y pausando el movimiento del robot.

Para lograr este objetivo, hemos diseñado e implementado diversas estructuras de datos y funciones. La clase `eCelda` define la estructura básica de una celda del mapa. Igualmente, la clase `eRobot` estructura los atributos necesarios para el control y seguimiento de los robots.

El proceso de inicialización del sistema se realiza a través de funciones dedicadas como `Init_Fig` e `Init_Mapa`. `Init_Fig` se encarga de cargar las imágenes necesarias para la representación gráfica del mapa y del robot utilizando la funcion ‘Load_Image’ ya vista en clases, mientras que `Init_Mapa` establece las características iniciales del mapa. La visualización del mapa se maneja mediante la función `Pinta_Mapa`.

El movimiento y la lógica de los robots se gestionan con la función `Mueve_Robot`, que implementa un algoritmo de control para dirigir a los robots a través del mapa. Esta función no solo controla el desplazamiento de los robots, sino que también supervisa el cumplimiento de los objetivos y condiciones establecidas para el correcto funcionamiento de las soluciones planteadas y finalización del recorrido.

Finalmente, la función `Final()` proporciona un mecanismo para concluir la simulación, mostrando el mensaje cuando el robot alcanza su destino y deteniendo el sistema de manera controlada.

# Explicación Código Mapa Inteligente.

## 1.0) Definición de constantes.

```
nRes = (640,640); nt_WX = nt_HY = 32; nMAX_ROBOTS = 1; lGo = True
nMx = nMy = 0; nR_1 = 610 ; nR_2 = 32
```

Aquí declaramos variables globales que usaremos y modificaremos más tarde, la resolución de la pantalla, el tamaño de los tiles, valores auxiliares y la bandera de nuestro ciclo principal.

## 1.1) Definición del objeto celda para la creación del mapa.

```
class eCelda(ct.Structure):
    _fields_ = [
        ('nT',ct.c_ubyte), # Tipo de Tile/Baldosa
        ('nS',ct.c_ubyte), # 0 : No se pinta - # 1 : Si se pinta
        ('nF',ct.c_ubyte), # Fila de Mapa
        ('nC',ct.c_ubyte), # Columna de Mapa
```

En esta Clase definimos la estructura y las variables de la celda, las cuáles son necesarias para definir los posibles estados en los que las celdas puedan estar, y así poder operar e iterar sobre ellos.

## 1.2) Definición del objeto robot.

```
#-----------------------------------------------------
#       Estructura Robots
#-----------------------------------------------------
class eRobot(ct.Structure):
    __fields__ = [
        ('nF',ct.c_ushort), #Figura
        ('nX',ct.c_ushort), #Pos X
        ('nY',ct.c_ushort), #Pos Y
        ('nR',ct.c_ushort), #Rango
        ('dX',ct.c_ushort), #dirección en x
        ('dY',ct.c_ushort), #dirección en Y
        ('nV',ct.c_ushort), # Velocidad
        ('nC',ct.c_ushort)  # número Columna
]
```

Cómo consiguiente tenemos también la función que ocuparemos para darle forma a nuestro robot, refiriéndonos a las variables de, ya sea, estado del robot, la

figura que este ocupará, los pasos, la posición y la velocidad, ya que serán importantes para el correspondiente movimiento del robot.

### 1.3) Iniciamos las imágenes

```
def Init_Fig():
    aImg = []
    aImg.append(Load_Image('T01.png',False )) # Tile base     id = 0
    aImg.append(Load_Image('T02.png',False )) # Tile amarillo id = 1
    aImg.append(Load_Image('Bo1.png',True  )) # Robot 1       id = 2
                                ...
    aImg.append(Load_Image('Bo8.png',True  )) # Robot 8       id = 9
    aImg.append(Load_Image('Rat.png',True  )) # Mouse 9       id = 10
    aImg.append(Load_Image('Msg.png',False )) # Mensaje final id = 11
    return aImg
```

Cargamos las imágenes necesarias en un arreglo para poder mostrarlas en pantalla cuando sea necesario.

### 1.4) Inicialización del robot y sus variables

```
def init_Robot():
    for i in range(0,nMAX_ROBOTS):
        aBoe[i].nF = 1    # Robot Tipo 1
        aBoe[i].nX = 0
        aBoe[i].nY = 0
        aBoe[i].nR = nR_1
        aBoe[i].nS = 1    # Switch por defecto
        aBoe[i].dX = 0
        aBoe[i].dY = 1  # el robot va hacia abajo
        aBoe[i].nV = 1
        aBoe[i].nC = 1
    return
```

Iniciamos nuestro robot definiendo todas las características necesarias de este, definiremos la imagen con la que iniciará (nF = 1 ‘Bo1.png’), que comenzará bajando (dY = 1 e dX = 0), también tendremos variables que ocuparemos más tarde para calcular su posición en X e Y o un estado como (nS = 1) sobre el cual operaremos para cambiar su dirección o nC el cual operaremos para cambiar la figura que se muestra en pantalla.

### 1.5) Visualización del robot

```
def Pinta_Robot():
    for i in range(0,nMAX_ROBOTS): # Iteramos las 8 Figuras del Robot
        if aBoe[i].nF == 1: sWin.blit(aFig[2] ,(aBoe[i].nX,aBoe[i].nY))
                                ...
        if aBoe[i].nF == 8: sWin.blit(aFig[9] ,(aBoe[i].nX,aBoe[i].nY))
    return
```

Esta función muestra en pantalla las diferentes figuras del robot, se irá mostrando en pantalla dependiendo del valor que tome nF.

### 1.6) Inicialización del mapa

```
def Init_Mapa():
    for nF in range(0,nRes[1] / nt_HY):
        for nC in range(0,nRes[0] / nt_WX):
            aMap[nF][nC].nT = 1 # inicializa el mapa con la tile amarilla
            aMap[nF][nC].nS = 0 # No se pinta por Defecto
            aMap[nF][nC].nF = nF # Fila de la Celda
            aMap[nF][nC].nC = nC # Colu de la Celda
    return
```

Para inicializar nuestro mapa tenemos los bucles necesarios para recorrer las dos dimensiones del mapa (Y,X) e ir definiendo el tipo nT = 1 (Tile amarilla ‘T02.png’) y las posiciones de todos los tiles, según corresponda.También definimos que la tile no se muestre en pantalla.

### 1.7) Creación del mapa

```
def Pinta_Mapa():

  for nF in range(0,nRes[1] / nt_HY):
     for nC in range(0,nRes[0] / nt_WX
      sWin.blit(aFig[0],(aMap[nF][nC].nC*nt_HY,aMap[nF][nC].nF*nt_WX))

            if  nC == (aBoe[0].nX+1)/nR_2 and nF == (aBoe[0].nY+1)/nR_2:
                if aMap[nF][nC].nT == 1:
                    aMap[nF][nC].nS = 1
            if aMap[nF][nC].nS == 1:
              sWin.blit(aFig[1],(aMap[nF][nC].nC*nt_HY,aMap[nF][nC].nF*nt_WX))
            else:
sWin.blit(aFig[0],(aMap[nF][nC].nC*nt_HY,aMap[nF][nC].nF*nt_WX))
```

Luego la función encargada de visualizar el mapa de manera correcta se puede analizar de la siguiente manera; declaramos doble bucle for para recorrer las dimensiones del mapa (matriz) en Y y en X respectivamente, estas se dividen en el ancho y alto de cada tile para obtener la cantidad de tiles de ancho y de alto y así mostrar en pantalla el tipo de tile 0 (Tile base ‘T01.png’) para cada tile.

Seguiremos con las sentencias condicionales que nos llevarán a mostrar el mapa; primero preguntamos la posición actual del robot, dividiendo su posición actual, a la cual le sumamos 1 para que cubra hasta la última fila y columna del mapa,entre el tamaño de los tiles, entonces, si es que la tile es la tile amarilla (nT = 1) nos encargamos de cambiar de cambiar el valor para que se pinte (nS = 1), a continuación preguntamos si la tile tiene el valor que corresponde a que se pinte, y si es así con la función de PyGame ‘.blit’ mostramos en pantalla la tile amarilla (‘T02.png’), si no mostramos en pantalla la tile base.

### 1.8) Lógica de movimiento del robot

```
def Mueve_Robot():
    for i in range(0,nMAX_ROBOTS): # Recorrimos todos los Robots
        aBoe[i].nR -= 1       # Decrementamos en 1 el Rango del Robot
        if aBoe[i].nR <= 0:    # Robot terminó sus pasos?
            if aBoe[i].nS == 1:
                aBoe[i].nS = 2  # Cambio de estado
                aBoe[i].nR = nR_2 # Robot ESTE nR_2 pasos
                aBoe[i].dX = 1 ; aBoe[i].dY = 0
            elif aBoe[i].nS == 2:
                aBoe[i].nS = 3  # Cambio de estado
                aBoe[i].nR = nR_1 # Robot SUBE nR_1 pasos
                aBoe[i].dX = 0 ; aBoe[i].dY = -1
            elif aBoe[i].nS == 3:
                aBoe[i].nS = 4  # Cambio de estado
                aBoe[i].nR = nR_2 # Robot ESTE nR_2 pasos
                aBoe[i].dX = 1 ; aBoe[i].dY = 0
            else:
                aBoe[i].nS = 1  # Cambio de estado
                aBoe[i].nR = nR_1 # Robot BAJA nR_1 pasos
                aBoe[i].dX = 0 ; aBoe[i].dY = 1
```

En esta función declaramos los movimientos del robot, primeramente recorremos los robots, luego tomamos el rango para disminuirlo en 1 comenzando desde nR (El cual lo definimos como = a nuestra variable global nR_1 = 610), entonces nuestro rango va desde 610 hasta 0 (el final del mapa), a continuación revisamos si el robot llegó al final del mapa para hacerlo cambiar de estado, comenzando en 1 lo volvemos 2 (para luego iterar sobre ese nuevo valor de nS), cambiamos nR en el rango de 32 pixeles hasta 0 (ancho del tile (nR_2 = 32)), luego cambiamos los valores de las direcciones en X e Y para que siga el movimiento deseado y así seguimos cambiando los pasos y direcciones (nR ,dX y dY) hasta que el estado de nS vuelve a ser uno y el ciclo se repite.

```
    #Actualizamos (Xs,Ys) de los Sprites en el Mapa 2D
    #-------------------------------------------------
       newX = aBoe[i].nX + aBoe[i].dX * aBoe[i].nV
       newY = aBoe[i].nY + aBoe[i].dY * aBoe[i].nV
```

Seguidamente guardamos la actualización de la posición del robot en unas nuevas variables auxiliares llamadas ‘newX’ y ‘newY’, las cuales serán; la dirección

actual de ese eje, más la dirección actual de ese eje multiplicando la velocidad del robot (nV = 1).

```
        if 0 <= newX < nRes[0] - nt_WX and 0 <= newY < nRes[1] - nt_HY:
            if newX == 607 and newY == 32:
                aBoe[i].nR = 0
                Final()
```

Luego nos aseguramos de que el robot esté dentro de ciertos límites para que este se mantenga en el mapa, para así preguntar si se encuentra en ciertas posiciones (x= 607, y=32) y este se detenga llamando a la función 'Final()'.

```
            else:
                aBoe[i].nX = newX
                aBoe[i].nY = newY
```

Si no está en esas posiciones simplemente guarda su posición actual en newX y newY.

```
        aBoe[i].nC += 1
        if aBoe[i].nC >= 20:
            aBoe[i].nC = 1
            aBoe[i].nF += 1
            if aBoe[i].nF == 9:
                aBoe[i].nF = 1
    return
```

Finalmente este fragmento de código realiza un seguimiento de las dos variables (nC y nF) del robot. Se incrementa en 1 'nC' por cada iteración del ciclo for y luego se verifica si ha alcanzado o superado 20. Si es así, 'nC', se reinicia a 1 y se incrementa 'nF', luego, si 'nF' alcanza 9, también se reinicia a 1.

### 1.9) Finalización del programa

```
def Final():

    sWin.blit(aFig[11],(200,200)) #Muestra la bandera
    pg.display.flip()#Actualiza la pantalla para mostrar la bandera
    global lGo #Llama a la variable global lGo
    while lGo:
        e = pg.event.wait()    #Pausa el juego hasta un evento quit
        if e.type in (pg.QUIT, pg.KEYDOWN):
```

```
lGo = False
```

La función ‘Final()’ muestra la bandera (‘Msg.png’) en la ventana, esto en la posición (200,200), para esto actualizamos la pantalla con la función de PYGAMES ‘.display.flip’ antes de llamar a la bandera del ciclo principal para así luego llamar al evento de pausa (‘.event.wait’’) el cual espera hasta que ocurra un evento de cierre de ventana (‘pg.QUIT’) o se presione una tecla cualquiera (‘pg.KEYDOWN’) y así cambiar el estado de la bandera del ciclo principal y terminarlo. Durante este tiempo, la ejecución del programa se pausa, y una vez que se registra uno de estos eventos, la función finaliza.

## Conclusión.

A través de las soluciones planteadas se ha demostrado la aplicación práctica del mapa inteligente.

La definición detallada de las clases `eCelda` y `eRobot` ha permitido establecer una base sólida para la manipulación de los tiles y del robot dentro del mapa inteligente. Las celdas, con sus atributos específicos como tipo, estado, fila y columna, y el robot, con sus coordenadas actuales, dirección, velocidad y estado, han sido fundamentales para gestionar las interacciones con los tiles y el movimiento dentro del mapa inteligente. Este enfoque estructurado ha facilitado el control preciso y la visualización del mapa y el robot en acción.

La implementación de funciones clave como `Init_Fig`, `Init_Mapa`, `Pinta_Mapa`, `Mueve_Robot`, y `Final` ha permitido crear un flujo de trabajo coherente y funcional para la simulación. La función `Init_Fig` ha sido crucial para la carga de imágenes necesarias para la visualización del mapa, del mensaje final y del robot. `Init_Mapa` ha establecido el mapa base con el tipo de celda definido y sus posiciones correspondientes. `Pinta_Mapa` ha sido responsable de la correcta visualización del mapa y la actualización del estado de los tiles en función de la posición del robot. `Mueve_Robot` ha manejado la lógica del movimiento del robot asegurando que sigan las rutas establecidas y actualizando sus posiciones de manera continua. Finalmente, la función `Final` ha proporcionado una forma de concluir la simulación implementando el mensaje solicitado cuando el robot alcanza su posición objetivo.

Este proyecto no solo ha cumplido con las problemáticas planteadas para este proyecto, sino que también nos ha ofrecido una plataforma práctica para experimentar con la programación y la visualización del mapa con los tiles y el control del robot.